Published in the Proceedings of the 23rd Annual Software Engineering Workshop, NASA Goddard Space Flight Center, Greenbelt, MD, December 2-3, 1998

Performing Verification and Validation in Reuse-Based Software Engineering

Edward A. Addy
NASA/WVU Software Research Laboratory

304-367-8353 (Voice)
304-367-8211 (Fax)
eaddy@wvu.edu

1. INTRODUCTION

The implementation of reuse-based software engineering not only introduces new activities to the software development process, such as domain analysis and domain modeling, it also impacts other aspects of software engineering. Other areas of software engineering that are affected include Configuration Management, Testing, Quality Control, and Verification and Validation (V&V). Activities in each of these areas must be adapted to address the entire domain or product line rather than a specific application system. This paper discusses changes and enhancements to the V&V process, in order to adapt V&V to reuse-based software engineering.

V&V methods are used to increase the level of assurance of critical software, particularly that of safety-critical and mission-critical software. Software V&V is a systems engineering discipline that evaluates software in a systems context [Wallace and Fujii 1989]. The V&V methodology has been used in concert with various software development paradigms, but always in the context of developing a specific application system. However, the reuse-based software development process separates domain engineering from application engineering in order to develop generic reusable software components that are appropriate for use in multiple applications.

The earlier a problem is discovered in the development process, the less costly it is to correct the problem. To take advantage of this, V&V begins verification within system application development at the concept or high-level requirements phase. However, a reuse-based software development process has tasks that are performed earlier, and possibly much earlier, than high-level requirements for a particular application system.

In order to bring the effectiveness of V&V to bear within a reuse-based software development process, V&V must be incorporated within the domain engineering process. Failure to incorporate V&V within domain engineering will result in higher development and maintenance costs due to losing the opportunity to discover problems in early stages of development and having to correct problems in multiple systems already in operation. Also, the same V&V activities will have to be performed for each application system having mission or safety-critical functions.

On the other hand, it is not possible for all V&V activities to be transferred into domain engineering, since verification extends to the installation and operation phases of development and validation is primarily performed using a developed system. This leads to the question of which existing (and/or new) V&V activities would be more effectively performed in domain engineering rather than in (or in addition to) application engineering. Related questions include how to identify critical reusable components, how to identify reusable components for which V&V at the domain level would be cost-effective, and how to determine the level to which V&V should be performed on the reusable components and on the domain architecture itself..

This paper discusses a framework for performing V&V within reuse-based software engineering that has been presented in [Addy 1998]. The framework identifies V&V tasks that could be performed in domain engineering, V&V tasks that could be performed in the transition from domain engineering to application engineering, and the impact of these tasks on application V&V activities. This paper further considers the extension of criticality analysis from an application-specific context to a product-line context.

2. DIFFERENCES BETWEEN V&V AND COMPONENT CERTIFICATION

Much work has been done in the area of component certification, which is also called evaluation, assessment, or qualification. These terms can have slightly different meanings, but refer in general to rating a reusable component against a specified set of criteria.

The common thread through these certification processes is the focus on the component rather than on the systems in which the component will eventually be (re)used. Dunn and Knight [1993] note that with the exception of the software industry itself, customers purchase systems and not components. Ensuring that components are well designed and reliable with respect to their specifications is necessary but not sufficient to show that the final system meets the needs of the user. Component evaluation is but one part of an overall V&V effort within reuse-based software engineering, analogous to code evaluation in V&V of an application system.

Another distinction between V&V and component certification is the scope of the artifacts that are considered. While component certification is primarily focused on the evaluation of reusable components (usually code-level components), V&V also considers the domain model and the generic architecture, along with the connections between domain artifacts and application system artifacts. Some level of component certification should be performed for all reusable components, but V&V is not always appropriate. V&V should be conducted at the level determined by an overall risk mitigation strategy.

3. FRAMEWORK FOR PERFORMING V&V WITHIN REUSE-BASED SOFTWARE ENGINEERING

One of the working groups at the 1996 Reuse Workshop (Reuse ’96) developed a framework for performing V&V within reuse-based software engineering [Addy 1996]. This framework is illustrated in Figure 1, and is described in more detail in [Addy 1998].

Domain-level V&V tasks are performed to ensure that domain products fulfill the requirements established during earlier phases of domain engineering. Transition-level tasks provide assurance that an application artifact correctly implements the corresponding domain artifact. Traditional application-level V&V tasks ensure the application products fulfill the requirements established during previous application life-cycle phases.

Performing V&V tasks at the domain and transition levels will not automatically eliminate any V&V tasks at the application level. However, it should be possible to reduce the level of effort for some application-level tasks. The reduction in effort could occur in a case where the application artifact is used in an unmodified form from the domain component, or where the

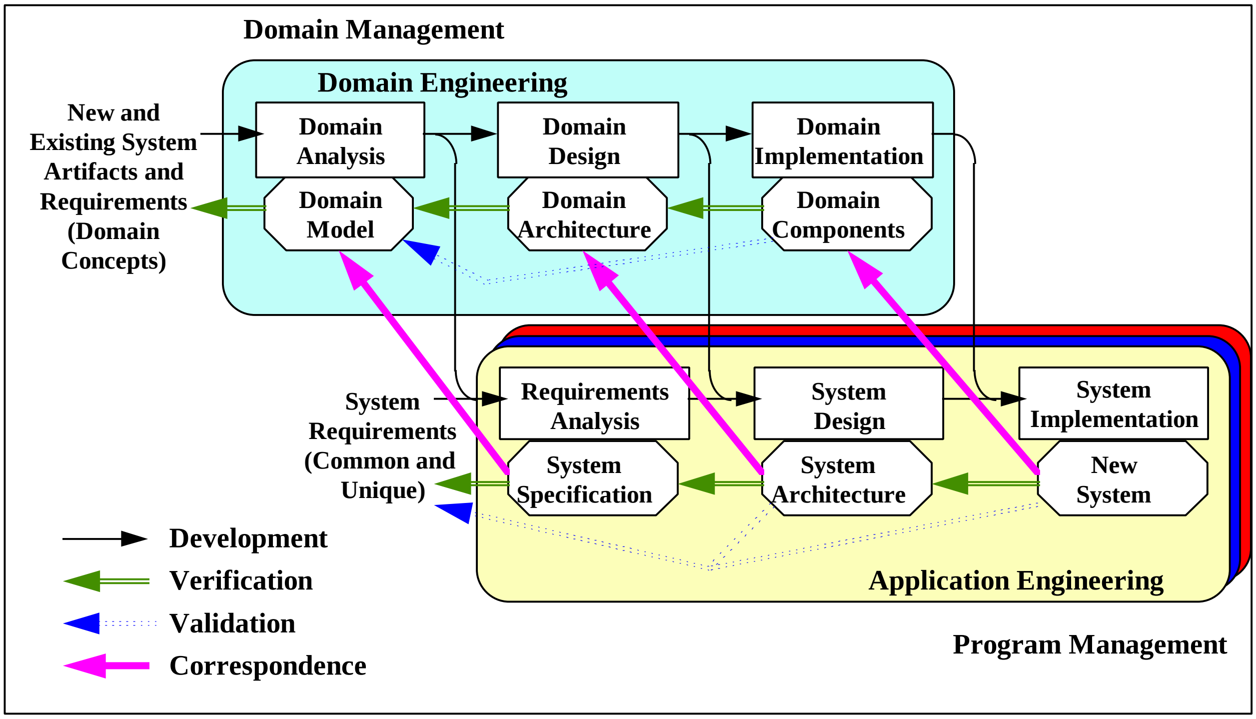

Figure 1: Framework for V&V within Reuse-Based Software Engineering

application artifact is an instantiation of the domain component through parameter resolution or through generation.

Domain maintenance and evolution are handled in a manner similar to that described in the operations and maintenance phase of application-level V&V. Changes proposed to domain artifacts are assessed by V&V to determine the impact of the proposed correction or enhancement. If the assessment determines that the change will impact a critical area or function within the domain, appropriate V&V activities are repeated to assure the correct implementation of the change.

4. V&V OF GENERAL COMPONENTS

The discussion in Section 3 focused on the issue of V&V within domain engineering, in the situation where the final systems would be subject to V&V even if the systems were not developed within a reuse environment. Many of the same justifications for performing V&V in a product line that includes critical systems also apply to V&V of other product lines and to general purpose reusable components. The Component Verification, Validation and Certification Working Group at WISR 8 found four general considerations that should be used in determining the level of V&V of reusable components [Edwards and Wiede 1997]:

- Span of application – the number of components or systems that depend on the component
- Criticality – potential impact due to a fault in the component
- Marketability – degree to which a component would be more likely to be reused by a third party
- Lifetime – length of time that a component will be used

The domain architecture serves as the context for evaluating software components in a product-line environment. However, this architecture may not exist for general use components. The Working Group determined that the concept of validation was different for a general use component than for a component developed for a specific system or product line. In the latter case, validation refers to ensuring that the component meets the needs of the customer. A general use component has not one customer, but many customers, who are software developers rather then end-users. Hence validation of a general use component should involve the assurance (and supporting documentation) that the component satisfies a wide range of alternative usages, rather than the specific needs of a particular end-user.

5. CRITICALITY ANALYSIS

Although not shown as a specific V&V task for any particular phase of the life-cycle, criticality analysis is an integral part of V&V planning. Criticality analysis is performed in V&V of application development in order to allocate V&V resources to the most important (i.e., critical) areas of the software [IEEE Std 1059-1993]. This assessment of criticality and the ensuing determination of the level of intensity for V&V tasks are crucial also within reuse-based software engineering.

The Criticality Analysis and Risk Assessment (CARA) method is one of the tools used by V&V practitioners at the NASA IV&V Facility to evaluate and prioritize the areas of risk within an application system. The CARA method includes a determination of both the likelihood that an error will occur in the software (risk) and the impact of an error occurring in that software (criticality). A team determines the criteria by which the software will be evaluated, which includes areas such as performance and operation, safety, development cost, and development schedule. Each software component receives a score based on these factors, and the scores are used to determine the level of V&V to be applied to each component. A CARA assessment is performed once with each major development milestone.

Other methods that are used to perform criticality analysis (or risk analysis) include the Software Engineering Institute Software Risk Evaluation Method and the NASA Continuous Risk Management process based on the SEI SRE, the NASA Ames Research Center approach to risk management within an ISO 9001 environment, the Jet Propulsion Laboratory System Risk Balancing method, the V&V Goal-Question-Metric process, and the SAIC Risk Cube method. [WORM 98] All of these methods share with CARA a tabular-based aggregation approach to determine overall risk, by evaluating fundamental risk factors and combining them in some fashion to determine aggregate risk of a system, subsystem, or component.

Criticality analysis methods need to be extended to include consideration of multiple application systems developed from reusable components. Each software component should be evaluated not only as is done with current methods, but also with consideration of the criteria listed in Section 4. For example, a component that does not score highly on any specific application system might require a high level of V&V because it is used (or anticipated to be used) in a large number of systems.

The current methods to perform analysis on software architectures [Tracz 1996, Garlan 1995] are also directed at the architecture for an application system rather than a product line architecture. One of the approaches being researched is a scenario-based analysis approach, Software Architecture Analysis Method [Kazman, et al. 1996], that could be extended to product line architectures. In the area of correspondence tasks, the Centre for Requirements and Foundations at Oxford is developing a tool (TOOR) to support tracing dependencies among evolving objects [Goguen 1996].

6. CONCLUSION

The framework for performing V&V in traditional application system development can be extended to reuse-based software engineering. The extended framework allows the V&V effort to be amortized over the systems within the domain or product line. Just as with V&V in application system development, V&V should be performed as part of an overall risk mitigation strategy within the domain or product line.

The primary motivation for V&V within domain engineering is to find and correct errors in the domain artifact in order to prevent the errors from being propagated to the application systems. This motivation is especially strong where the application systems perform critical functions. Even if there are no critical functions performed by the systems within the domain, V&V might be appropriate for a component that has the potential to be used in a large number of application systems.

ACKNOWLEDGEMENT

This work is funded by NASA Cooperative Agreement NCC 2-979 at the NASA/Ames IV&V Facility in Fairmont, WV.

REFERENCES

Addy, Edward A. (1998), "A Framework for Performing Verification and Validation in Reuse-Based Software Engineering," Annals of Software Engineering, Vol. 5, 1998.

Addy, Edward A. (1996), "V&V Within Reuse-Based Software Engineering", Proceedings for the Fifth Annual Workshop on Software Reuse Education and Training, Reuse '96, http://www.asset.com/WSRD/conferences/proceedings/results/addy/addy.html.

Dunn, Michael F. and John C. Knight (1993), "Certification of Reusable Software Parts," Technical Report CS-93-41, University of Virginia, Charlottesville, VA.

Edwards, Stephen H. and Bruce W. Wiede (1997), "WISR8: 8th Annual Workshop on SW Reuse", Software Engineering Notes, 22, 5, 17-32.

Garlan, David (1995), "First International Workshop on Architectures for Software Systems Workshop Summary", Software Engineering Notes, 20, 3, 84-89.

Goguen, Joseph A. (1996), "Parameterized Programming and Software Architecture," In Proceedings of the Fourth International Conference on Software Reuse, IEEE Computer Society Press, Los Alamitos, CA, pp. 2-10.

IEEE STD 1059-1993, IEEE Guide for Software Verification and Validation Plans, Institute of Electrical and Electronics, Inc., New York, NY.

Kazman, Rick, Gregory Abowd, Len Bass, and Paul Clements (1996), "Scenario-Based Analysis of Software Architecture," IEEE Software, 13, 6, 47-55.

Tracz, Will (1996), “Test and Analysis of Software Architectures,” In Proceedings of the International Symposium on Software Testing and Analysis (ISSTA ’96), ACM Press, New York, NY, pp 1-3.

Wallace, Dolores R. and Roger U. Fujii (1989), Software Verification and Validation: Its Role in Computer Assurance and Its Relationship with Software Project Management Standards,” NIST Special Publication 500-165, National Institute of Standards and Technology, Gaithersburg, MD.

WORM 98, Proceedings of the Workshop on Risk Management, October 1998.